



@ngrx/store

Using HttpClient Directly



Peter Kassenaar
info@kassenaar.com

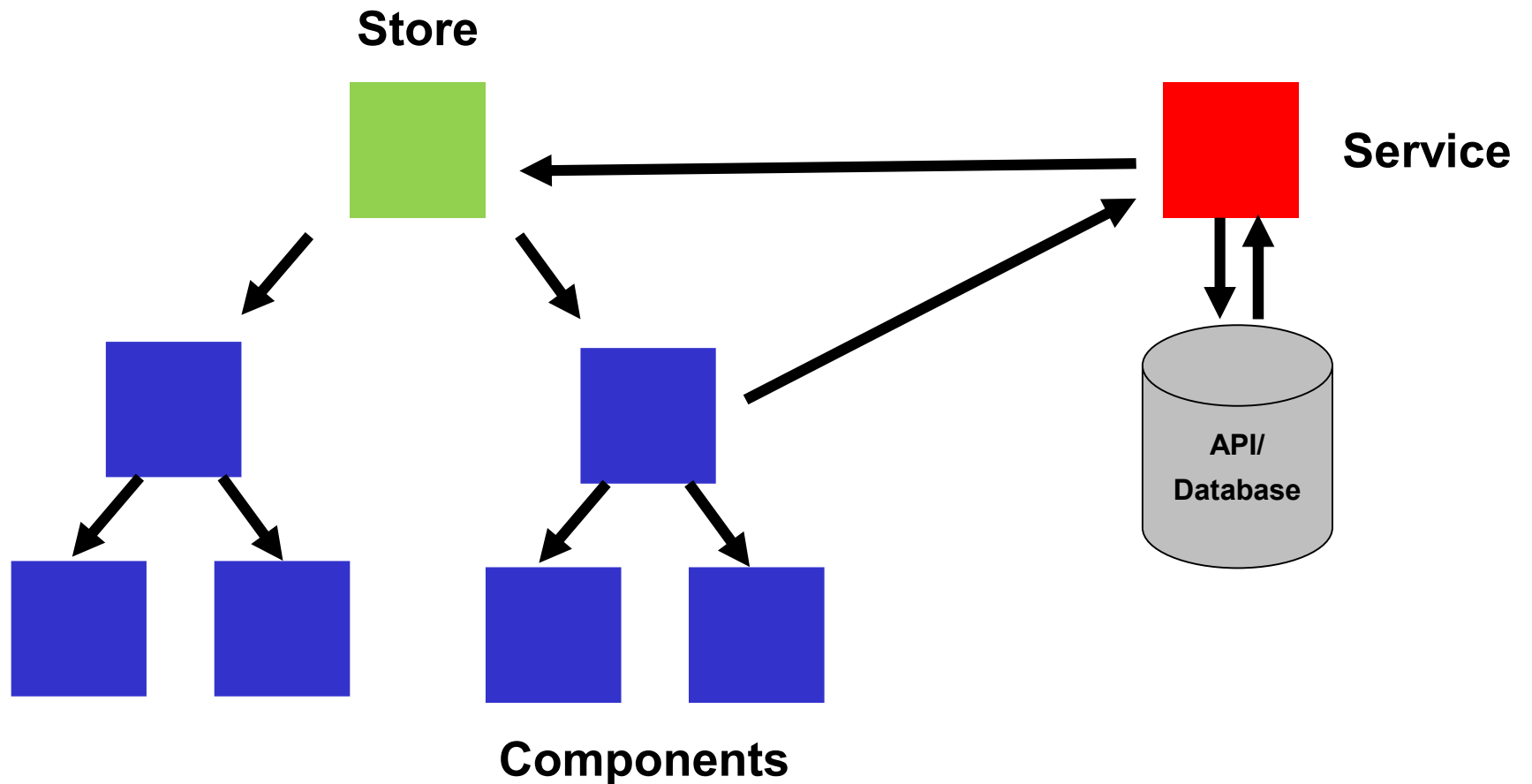


Using HttpClient directly

Talking RESTful to real API's – plain and simple!

Architecture

Call API in Service, dispatch to Store, subscribe in Components

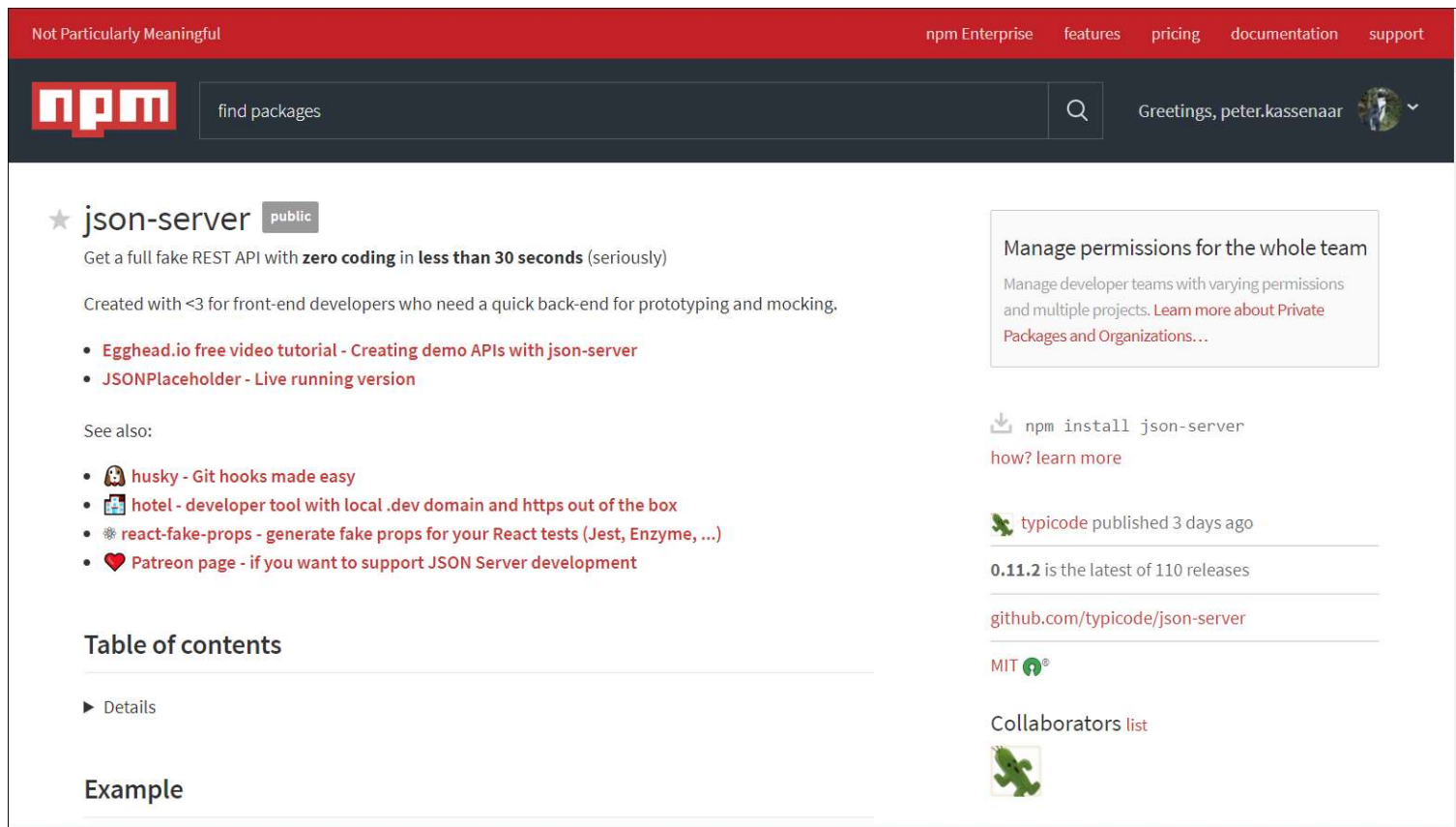


Actions and Reducers

- **No changes** on Actions and reducers.
- Add a **service** (if you haven't done so already) that talks to the outside world
- When a result comes back, **dispatch** the result to the store.

First – add a server

- We're using `json-server` here
- Provides a simple RESTful API, based on `.json`-file in webroot



The screenshot shows the npm website interface for the `json-server` package. At the top, there's a red navigation bar with links to 'npm Enterprise', 'features', 'pricing', 'documentation', and 'support'. Below this is a dark search bar with the 'npm' logo and a search icon. The main content area displays the package details for `json-server`, which is marked as 'public'. The description states it's a 'full fake REST API with zero coding in less than 30 seconds (seriously)'. It also mentions it was created with <3 for front-end developers. A list of related resources includes a video tutorial on Egghead.io and a live running version on JSONPlaceholder. A 'See also' section lists other tools like husky, hotel, react-fake-props, and a Patreon page. On the right, there's a sidebar with a 'Manage permissions' section, a code snippet for installing the package, a link to learn more, and information about the latest release (0.11.2) and the repository on GitHub. The bottom of the page shows a 'Table of contents' with links to 'Details' and 'Example'.

Not Particularly Meaningful

npm Enterprise features pricing documentation support

npm find packages

Greetings, peter.kassenaar

★ json-server public

Get a full fake REST API with **zero coding** in **less than 30 seconds** (seriously)

Created with <3 for front-end developers who need a quick back-end for prototyping and mocking.

- [Egghead.io free video tutorial - Creating demo APIs with json-server](#)
- [JSONPlaceholder - Live running version](#)

See also:

- [husky - Git hooks made easy](#)
- [hotel - developer tool with local .dev domain and https out of the box](#)
- [react-fake-props - generate fake props for your React tests \(Jest, Enzyme, ...\)](#)
- [Patreon page - if you want to support JSON Server development](#)

Table of contents

- Details

Example

Manage permissions for the whole team

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

npm install json-server

[how? learn more](#)

[typicode](#) published 3 days ago

0.11.2 is the latest of 110 releases

[github.com/typicode/json-server](#)

MIT

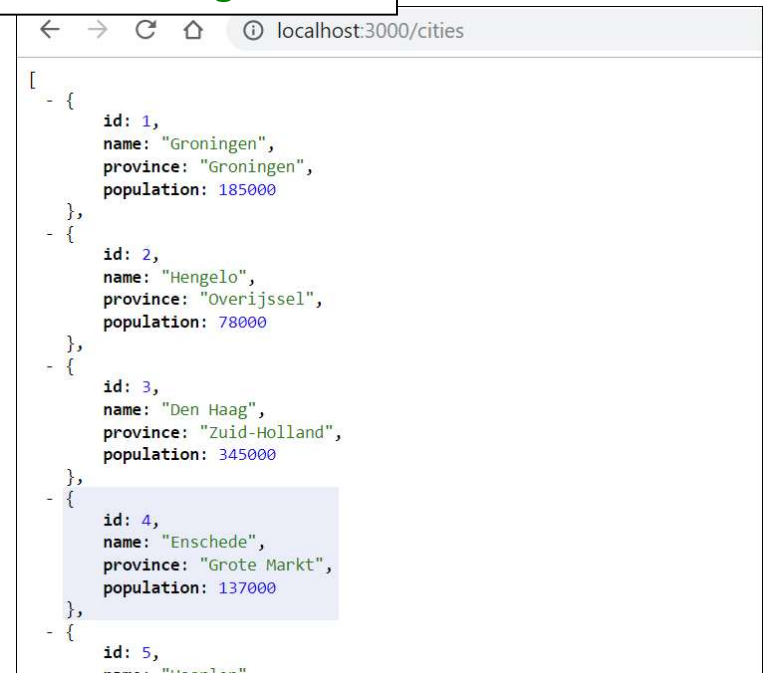
Collaborators [list](#)

<https://www.npmjs.com/package/json-server>

Add a script to start json-server

- NOT necessary if you talk to a 'real' endpoint.
- But we're using `cities.json` and `json-server` here. So add to `package.json`

```
"json-server": "json-server --watch cities.json"
```



A screenshot of a web browser window displaying a JSON array of city data. The browser's address bar shows 'localhost:3000/cities'. The JSON data is as follows:

```
[  
  - {  
    id: 1,  
    name: "Groningen",  
    province: "Groningen",  
    population: 185000  
  },  
  - {  
    id: 2,  
    name: "Hengelo",  
    province: "Overijssel",  
    population: 78000  
  },  
  - {  
    id: 3,  
    name: "Den Haag",  
    province: "Zuid-Holland",  
    population: 345000  
  },  
  - {  
    id: 4,  
    name: "Enschede",  
    province: "Grote Markt",  
    population: 137000  
  },  
  - {  
    id: 5,  
    name: "Heerlen"
```

Add HttpClientModule to application

- Update `app.module.ts` and `city.service.ts`
- Since we're using services, the HTML and Component are unaltered
- Use `HttpClientModule` in Module and Service

```
import {http[Client]Module} from '@angular/common/http';
...

@NgModule({
  ...
  imports      : [
    HttpClientModule,
  ],
  ...
})
```

Edit city.service.ts

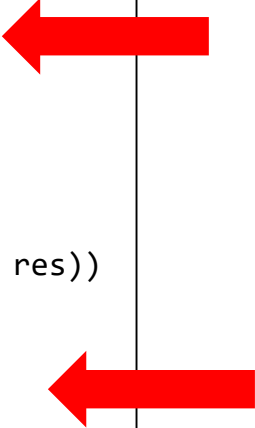
Add `Http` and call API in `loadCities()`.

Upon subscription, dispatch data to the store

```
// Some stuff that our server (json-server) needs:
const BASE_URL = 'http://localhost:3000/cities';
const HEADERS = {
  headers: new HttpHeaders().set('Content-Type', 'application/json')
};

@Injectable({ providedIn: 'root' })
export class CityService {
  constructor(private store: Store<CitiesState>,
              private http: HttpClient) {
    this.loadCities(); // load cities once the service is started
  }

  loadCities() {
    this.http.get(BASE_URL)
      .pipe(
        tap(res => console.log('We talked to json-server and received: ', res))
      )
      .subscribe((response: City[]) => {
        return this.store.dispatch(loadCities({cities: response}));
      });
  }
}
```



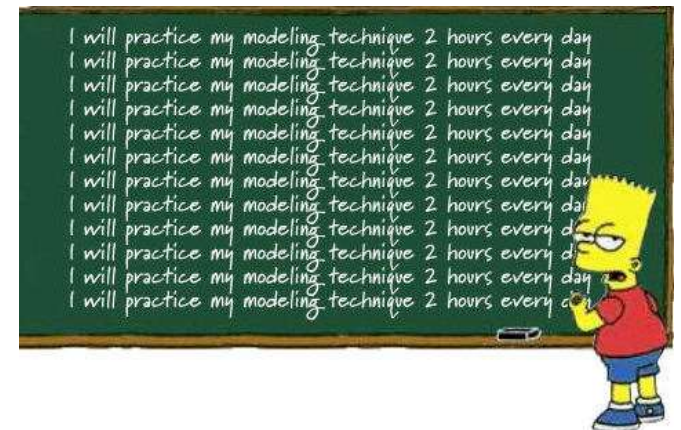
Adding and deleting cities

- Same procedure...

```
removeCity(city: City) {  
  this.http.delete(BASE_URL + `/${city.id}`, HEADERS)  
    .subscribe(() => {  
      console.log('City removed', city);  
      // optimistic delete - assume everything went fine in the backend,  
      this.store.dispatch(removeCity({city}));  
    });  
}  
addCity(city: City){  
  ...  
}
```

Workshop

- Use your own app, add a service and call HTTP to load .json-data
- OR: Start from `../215-ngrx-store-http`
- Make yourself familiar with the store concepts and http-flow. Study the example code.
- Add the `addCity()` method on the service, that adds a city to the .json file via json-server
- Add the `updateCity()` method on the service, to edit an existing city



Next Steps

- [@ngrx/effects](#) - Side Effect model for @ngrx/store to model event sources as actions.
- [@ngrx/router-store](#) - Bindings to connect the Angular Router to @ngrx/store
- [@ngrx/store-devtools](#) - Store instrumentation that enables a powerful time-travelling debugger
- [@ngrx/entity](#) - Entity State adapter for managing record collections.

<https://ngrx.io/docs>